Automatic High-Level Hardware Checkpoint Selection for Reconfigurable Systems

Alban Bourge, Olivier Muller, Frédéric Rousseau

Automatic High-Level Hardware Checkpoint Selection for Reconfigurable Systems

Alban Bourge, Olivier Muller and Frédéric Rousseau
Univ. Grenoble Alpes
TIMA Laboratory
Grenoble, France F-38031
{alban.bourge,olivier.muller,frederic.rousseau}@imag.fr

***Abstract*—Modern FPGAs provide great computational power and flexibility but there is still room for improving their performances. For example multi-user approaches are particularly underdeveloped as they require specific mechanisms still to be automated. Sharing an FPGA resource between applications or users requires a context switch ability. The latter enables pausing and resuming applications at system demand. This paper presents a method that automatically selects a good execution point, called hardware checkpoint, to perform a context switch on an FPGA. The method relies on a static analysis of the finite state machine of a circuit to select the checkpoint states. The obtained selection ensures that the context switch mechanism respects a given latency and tries to minimize the mechanism costs. The method takes advantage of its integration in an open-source HLS tool and preliminary results highlight its efficiency.**

***Index Terms*—FPGA, HLS, CAD, hardware context switch**

I. Introduction

In order to gain flexibility, FPGAs could greatly benefit from allowing multi-user utilization, or time-multiplexing [1][2], which enables to share one or multiple reconfigurable devices between several users or tasks. Efficient and generic time-multiplexing techniques are lacking in current FPGA design flows. Existing techniques rely on FPGA-specific or user-based solutions (cooperative multitasking paradigm). To make the share seamless for users and designers, a technology-independent preemptive paradigm should be adopted.

For reconfigurable systems, it implies to make an image of the state of the running circuit (the context), to save this description and to enable a restoration of the circuit in a previously saved state. For CPU-based machine, this functionality is known as *context switch*.

An ideal hardware context-switch mechanism ensures an answer to system preemption demands in a given latency and is FPGA-technology independent. The hardware overhead has also a negligible impact on the circuit performance and the memory footprint on the whole system should be minimized. Eventually the implementation and usage of the mechanism must be effortless for designers and users.

This is a complex and multi-faceted problem of which we will initiate an answer. The contributions of this paper are the following. We propose the specification of a hardware context switch mechanism introduced at a high level of abstraction. This specification intends to tackle all the previous points. Then, an open-source tool implementing the solution is revealed. To the best of our knowledge, the specified method has no precedent.

After giving few definitions and presenting the context of the problem in Section II, Section III will introduce the method and its benefits. The results of the proposed implementation are given in Section IV.

II. Context and definitions

A. Definitions

1) System: A hardware resource allowing context switch is constituted of a controlling system (e.g. a CPU supplied with an operating system) communicating with the FPGA. The FPGA is a slave for the OS, the latter activating context switch operations on the hardware resource.

2) Hardware context: Throughout the paper, it is assumed that the hardware task can be represented following an FSM/datapath model. Each state of the task represents actions on the datapath and has a set of *live* variables, i.e. memory elements accessed (read or written) during the state actions or in following states. It is assumed that the context of a task consists in the set of live variables of the current state and the initial configuration of the FPGA. The controlling system knows which configuration bitstream corresponds to the running task, hence it is assumed that saving this part of the context is already done. Conversely, the evolution of the memory elements is not predictable and a mechanism in order to read the memory elements back when a context switch occurs must be found. Memory elements in an FPGA are of three types: flip-flops, BRAM and LUTRAM.

3) Hardware checkpoint: We define a hardware checkpoint as follows: a state of a task where context switch operations are allowed. The context restoration restarts the task where it was stopped by using the previously saved hardware checkpoint.

4) Hardware context-switch: Figure 1 presents an example of a context switching process. Two preemption demands can be observed: the first switches task 1 with task 2 and the second switches back to task 1. Task 1 preemption is a two-step process represented by the time needed by the task to reach a checkpoint (*cp* step with total time *tc*) and the context saving of the task (*save* step with a duration of *ts*). *ts* varies according to the size of the context needing to be extracted. This saving step is followed by a configuration of the chip in order to launch task 2, which is preempted in its turn.

Finally, when the system wants to restore task 1, it has to add a restoration stage (*rest*) in order to re-write the previous context before resuming the execution of the first task.

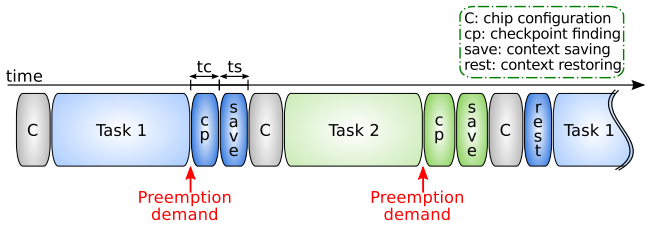
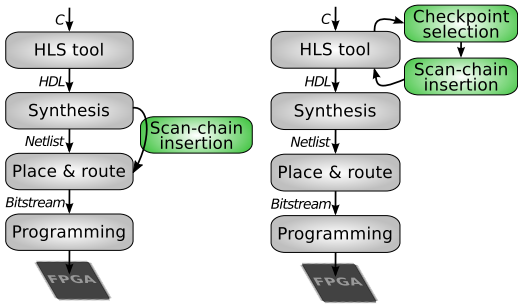


Fig. 1: Hardware context switch timeline



(a) Typical flow (b) Proposed flow

Fig. 2: Comparison between two design flows

B. Context extraction methods

Two main families exist in hardware context extraction techniques. The first consists in extracting the state of the FPGA components by the same mean used for configuration. Also called a *readback* method as the configuration, hence the flip-flop values, are read and transferred through the configuration mechanism back to the controlling system. [3] reports less than 8% of useful data as only memory elements values are needed. To cope with this limitation [4] filters the bitstream off-line and [3] reads back only used frames.

The second type directly add to the circuit structures enabling context extraction. The most common is a serial link of one or several bit wide between each flip-flop called a *scan-chain*. The extraction time and memory footprint are improved compared to readback method (with a similar width of bus and operating frequency) because only memory elements values are extracted. On the other hand, the extra design efforts needed to add such structures can be costly. Finally, the area overhead of such structure has to be taken into account. [5] describes three methods and put forward a tool automating design steps. [6] introduced *switching points* limiting switching states number but did not proposed a selection method. An example for both families is described and tested in [7].

III. Contributions

A scan-chain based solution is selected to address the problem of context extraction. This paragraph explains the whole mechanism and the two contributions made in order to reduce the limitations associated with scan-chain usage.

A. Hardware context switch mechanism

Preempting a hardware task requires the extraction of its context, which can represent a consequent amount of data. Being able to extract these data implies to introduce a hardware overhead to the circuit. Yet, the volume of data effectively requiring extraction will vary over the time. Thus, the hardware overhead can be limited by allowing extraction only at instants where the context volume is small. Indeed, we consider that a lighter context volume necessitates a lighter extraction mechanism. These instants are called hardware checkpoints. The method relies on a high-level analysis allowing to select hardware checkpoints for their properties.

Two main steps are applied to a high-level description of the selected hardware task. First, the hardware checkpoint selection is run. During this time, different partial scan-chains are identified. A partial scan-chain corresponds to the context associated with a state in opposition to a full scan-chain which is a chain between each flip-flops of a circuit. Secondly, the partial extraction mechanism is described in a high-level representation and added to the application. The Figure 2 illustrates a comparison between the typical design flow and the proposed design flow. In the first case, in Figure 2a, the scan-chain is inserted after logic synthesis. In our case, it is automatically embedded in the HDL layer (Figure 2b).

B. Checkpointing hardware tasks

The checkpoint selection step has a direct impact on the mechanism area and on the memory footprint. Both can be optimization objectives. One has globally a good impact on the other (e.g. limiting the size of the partial scan-chains is also leading to a better memory footprint). The time set by the controlling system known as *cs_latency* represents the latency constraint for context switching. For instance if a preemption demand raises in a random state, a checkpoint has to be reached (corresponding to time t_c) before context extraction (corresponding to time t_s) and the addition of these durations has to be inferior to the constraint. In that case, it is said that the checkpoint covers the states which led to it. With this definition, it is possible to pinpoint a certain set of checkpoints covering the entire circuit. Two algorithms are proposed. First, an analysis of the FSM identifies the coverage of each states. Secondly, a greedy heuristic is used to get a fast resolution of the NP-complete selection problem ("set covering problem" [8]) consisting in finding a set of checkpoint covering all the states and satisfying optimization objectives.

1) Circuit analysis: Analysing the FSM of the circuit, it is possible to retrieve the coverage of each states but complex situations has to be handled (loops, switchs ...). The Figure 3 illustrates how a switch is covered considering that each state lasts one clock cycle, $cs\_latency = 12$ cycles and state 7 has 8 bits needing extraction (i.e. $ts = 8$ cycles with a 1-bit scan-chain). The constraint being $tc + ts < cs\_latency$, the time left for reaching state 7 is at most $tc = cs\_latency - ts$

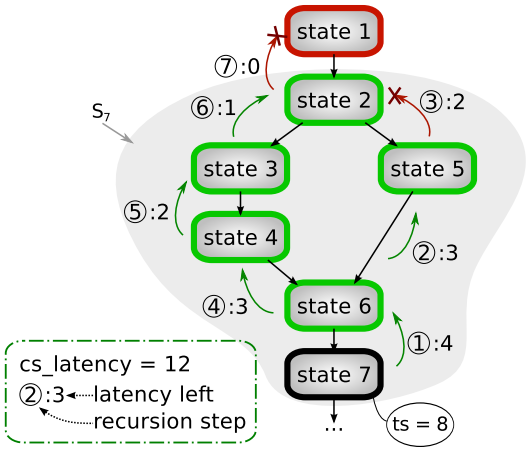


Fig. 3: Example of coverage computing for state 7

which gives $tc = 4$ clock cycles. The recursion hence starts from state 7 with 4 cycles left. Step ① is testing state 6 with the time left (4 cycles). The result of the test is positive (i.e. 7 covers 6) because the latency left is not null. Step ② is testing the previous states of 6, firstly 5. The latency left being positive, 5 is covered. Step ③ is not completed because 2 can not be covered if all the switch branches starting from it are not covered. Hence, the recursion starts back at state 6 with step ④. The second branch of the switch being covered in its turn, step ⑥ is completed and state 2 is covered. State 1 is not covered because no latency is left for step ⑦. The result of the coverage computing gives that states $\{2, 3, 4, 5, 6, 7\}$ are covered.

2) Greedy heuristic: Given the complexity of the minimization problem (NP-complete) a greedy heuristic is chosen to search the set of checkpoints. In [9], a greedy algorithm is proposed to solve the set covering problem with a linear minimization objective. This algorithm has a $O(n^2)$ complexity. It also ensures that the obtained solution is close from the minimal solution in a *a priori* defined factor.

At the end of the heuristic is obtained a set of checkpoints reducing the area overhead of the partial extraction mechanism and ensuring the task can reach a checkpoint within time $cs\_latency$. Though non optimal by nature, the next section shows that the heuristic gives good results.

C. High Level method

After the checkpoint selection, future addition of the structures enabling context switching operations will still be done at high-level. In the case of a basic design flow, the scan-chain addition occurs typically at the HDL or netlist level (cf. Figure 2a). Hence, efforts should be made in order to introduce the functionality inside an application. The proposed method removes all extra design efforts necessary to implement the partial extraction mechanism thanks to its integration into a HLS tool. It also allows the core algorithm to access all necessary data, such as live variables and $cs\_latency$. Moreover, the proposed method is entirely platform independent, as the context-switch mechanism is integrated in the application description. Finally, it allows an abstraction of the memory elements whatever the underlying implementations (flip-flop, BRAM or LUTRAM). As proposed in [10], it is possible to join each memory elements in a scan-chain, admitting the addition of specific mechanism for RAMs.

IV. Implementation and Results

This section presents the results obtained with our method on a set of typical applications. The method is fully integrated in a tool which currently support flip-flop extraction.

A. CP3

CP3, standing for *CheckPoint PinPoint*, implements the method described previously. It is written in C, free, open source and integrated as plugin in the HLS tool AUGH [11]. AUGH is designed for automatic generation of hardware accelerators for FPGA under resource constraints. The latter is also a free open source software and can be found at [12]. This tool produces a VHDL description from an input application written in C, independently from the targeted board or family of FPGA (Xilinx, Altera ...).

AUGH loads C applications to perform a design space exploration and applies transformations to the description. Then, CP3 is loaded and the hardware checkpoint search can start. The current result of CP3 is a set of checkpoints and an estimated cost of the associated partial extraction mechanism.

B. Results

We ran the tool CP3 on a set of typical applications to highlight our method efficiency. Six are part of the CHStone suite (adpcm, aes, blowfish, gsm, mpeg2 and sha) and we added an mjpeg decoder and an idct algorithm.

In the following experiments, the $cs\_latency$ value given is high enough (approx. 5000 clock cycles) to reach a stable algorithm output. Indeed, setting it higher is not modifying the obtained set of checkpoints. The target is a XC7V585T@100MHz of the Virtex 7 family. The type of chip is demanded by AUGH to determine the maximum area available for the design space exploration and to perform the mapping. In fact, any FPGA could have been used (Xilinx, Altera etc.) thanks to the level of abstraction of the method. As said previously, CP3 will provide a set of checkpoints and an estimation of the resulting overhead. The results are shown in Table I. A first observation can be made: the number of checkpoints is low compared to the number of states in the applications (geomean of 4.2%). The advantages are analyzed in following subsections.

1) Area overhead reduction: A significant reduction of the partial extraction mechanism size for each application compared to the full scan-chain method is achieved. The results are though very widely spread, starting from 17% (mpeg2) and reaching 96% (idct). Globally, the results show a 52% mean gain of the size of partial extraction mechanism compared to a full scan-chain.

No logic syntheses were run on the complete mechanism as the implementation is left for future work. However, a worst-case estimation is possible if we consider we can put two

TABLE I: Results of the proposed method on a common application set

			extraction mechanism size			partial extraction mechanism footprint		
	# checkpoints /states	checkpoint ratio	full scan-chain (bit)	partial extraction mech. (bit)	**gain**	max partial scan-chain (bit)	mean partial scan-chain (bit)	**gain (mean /full)**
adpcm	8/141	5.7 %	6304	4384	**30%**	4288	1844	**71%**
aes	7/1059	0.66 %	1616	200	**88%**	168	162	**90%**
blowfish	6/117	5.1 %	728	424	**42%**	224	108	**85%**
gsm	18/373	4.8 %	912	448	**51%**	160	54	**94%**
idct	3/238	1.3 %	1000	40	**96%**	40	34	**97%**
mjpeg	111/958	11.6 %	5826	2114	**64%**	770	444	**92%**
mpeg2	98/418	23.4 %	968	800	**17%**	448	376	**61%**
sha	9/298	3.0 %	3072	288	**91%**	160	96	**97%**
Geometric mean		4.2%			**52%**			**85%**

multiplexor in a LUT-6. AUGH also evaluates the area of the circuit, so we will use this value for comparison as can be seen in Table II. Our estimation gives for the aes application an addition of 488 multiplexors for creating the scan-chain between memory elements, which means 244 more LUT-6. This is extremely low compared to the approximately 100,000 LUT of our application. On the contrary, the adpcm and mpeg2 appear costly. For most applications, this worst-case overhead is affordable.

TABLE II: LUT-6 estimation for AUGH and CP3 overhead

	AUGH	CP3
adpcm	19443	6352 **(+33%)**
aes	110324	244 **(+ 0%)**
blowfish	18535	244 **(+ 1%)**
gsm	31984	400 **(+ 1%)**
idct	9513	36 **(+ 0%)**
mjpeg	291583	10307**(+ 4%)**
mpeg2	8662	6320 **(+73%)**
sha	34327	288 **(+ 1%)**
Geometric mean		(+2.4%)

2) Data footprint reduction: Compared to the frame-related readback method seen in [3], the gain in term of memory footprint represents orders of magnitude. Indeed, if we take an aes application, they obtain a readout of 21,952 bytes where we have a maximum of 624 bits (78 bytes). The comparison of both implementations is not trivial but it is to our disadvantage: our application uses approximately four times more memory elements.

Finally, the brightest example of memory footprint reduction is the mpeg2 decoder. When finding a set of checkpoints, CP3 is not reducing much the partial extraction mechanism size compared to the full scan-chain. On the contrary, when looking at partial scan-chains, it is still able to achieve a memory footprint reduction of 61%. More generally, the results are excellent with partial scan-chains since the geometric mean footprint reduction is around 85% compared to a full scan-chain approach.

3) Impact on developer: On the same set of applications a geometric mean execution time of $0.26\,\mathrm{s}$ is obtained. It represents an average of 5% of total AUGH execution time. Besides not involving the developer in the checkpointing selection, this method has no impact at all on the development time.

V. Conclusion

This paper presents a first step toward an automatic method to enable context switch on hardware tasks executed on reconfigurable systems. This universal method is based on the usage of hardware checkpoints to reduce its utilization impact. The automatic selection method has been integrated successfully in a demonstration tool named CP3 and inserted in a HLS design flow. Compared to traditional approaches, the obtained results confirm a limitation of the area overhead and a very small memory footprint, while having a negligible impact on development time.

The short term perspectives are to introduce effectively the scan-chain and add hardware in order to retrieve application state outside the FPGA, still at high level of abstraction.

References

[1] S. Trimberger *et al.*, "A time-multiplexed fpga," in *FCCM, 1997*. IEEE, 1997, pp. 22–28.

[2] S. M. Scalera and J. R. Vazquez, "The design and implementation of a context switching FPGA," in *FCCM, 1998*. IEEE, 1998, pp. 78–85.

[3] H. Kalte and M. Porrmann, "Context saving and restoring for multitasking in reconfigurable systems," in *FPL, 2005*. IEEE, 2005.

[4] H. Simmler *et al.*, "Multitasking on fpga coprocessors," in *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*. Springer, 2000, pp. 121–130.

[5] D. Koch *et al.*, "Efficient hardware checkpointing: concepts, overhead analysis, and implementation," in *Proceedings of the 2007 ACM/SIGDA*. ACM, 2007.

[6] J.-Y. Mignolet *et al.*, "Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip," in *DATE, 2003*. IEEE, 2003, pp. 986–991.

[7] K. Jozwik *et al.*, "Comparison of preemption schemes for partially reconfigurable FPGAs," *Embedded Systems Letters, IEEE*, vol. 4, no. 2, pp. 45–48, 2012.

[8] T. H. Cormen *et al.*, *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001, vol. 2.

[9] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[10] T. Wheeler *et al.*, "Using design-level scan to improve FPGA design observability and controllability for functional verification," in *FPL, 2001*. Springer, 2001, pp. 483–492.

[11] A. Prost-Boucle *et al.*, "Fast and standalone design space exploration for high-level synthesis under resource constraints," *Journal of Systems Architecture*, vol. 60, pp. 79–93, 2014.

[12] A. Prost-Boucle, "Augh project," 2013. [Online]. Available: http://tima.imag.fr/sls/research-projects/augh/